UDC 004.051

AN EFFECTIVE STRUCTURE FOR NEW DATA INTERCHANGE FORMAT

A. PRANKO, R. BOHUSH
Polotsk State University, Belarus

Most modern web applications use two JSON and XML data structures for data transfer. They both have a lot of advantages, among which:

• Code readability.
• Ease of creating a data object on the server side.
• Ease of processing data on the client side.
• Easy to expand.
• Debugging and correction of errors.

However, it is worth noting that JSON is used much more often in new applications, XML received a large part of the audience at a time when it was at the peak of its popularity. JSON has taken the lead, crowding out XML more and more for the following reasons:

• JSON has a simpler grammar, only 30 patterns versus 90 for XML, which provides a simpler implementation and further debugging.
• JSON has significantly less code redundancy. For example, two arrays are shown in Table 1 with lists storing the same information:

Table 1. – Comparing JSON and XML volume

JSON	`{`
	`  "users": [`
	`    {`
	`      "name": "Alice",`
	`      "age": 20`
	`    },`
	`    {`
	`      "name": "Bob",`
	`      "age": 46`
	`    }`
	`  ]`
	`}`

XML	`<users>`
	`  <user>`
	`    <name>Alice</name>`
	`    <age>20</age>`
	`  </user>`
	`  <user>`
	`    <name>Bob</name>`
	`    <age>46</age>`
	`  </user>`
	`</users>`

In the Table 1, the main drawback of XML is clearly visible compared to JSON, all its field names are duplicated in the opening and closing tags, and in the array, each element is additionally wrapped in an additional tag.

This redundancy of XML was the main reason for switching to JSON. However, JSON also has enormous redundancy. The format has a lot of unnecessary designs and here are the most obvious ones:

• Keywords may be repeated many times.
• Keywords are in double quotation marks.

• The colon and the value separate the keyword.
• If this is not the end of the object, then you need to set a comma after the value.
• Arrays and objects are surrounded by brackets.
• Lines must also be inside quotation marks.
• Numbers are written in a string, which also adversely affects their size.

It is also worth noting that JSON and XML have a readable form only in a formatted version, when transferred, they are cleaned of extra spaces and hyphens, losing this quality.

Considering all these shortcomings, a new data structure was designed. The structure is represented in binary form.

To solve the problem of repeatability of keywords for each, a unique id of two bytes will be set and the entire list of keywords will be located at the beginning of the message. Before each word, its id of two bytes and the length (also 2 bytes) of the keyword in bytes will be written in order to know which piece of the message to take next as a keyword. Before all this list there will be the first 2 bytes storing the total length of the list of keywords. Thus, long key glories will be recorded only once, and then in the main structure they will be recorded using id which occupies only 2 bytes.

Next, after the list of keywords, the main data structure will go. Any data will begin with its type occupying 1 byte. In the structure, I highlighted the following data types:

• Object - after the type, the total number of bytes that this object occupies should be indicated. This value will occupy 2 bytes. And then the data inside the object will go as a list. All of them will have a key id in front of their type.

• Array - repeats the structure of the object, only you do not need to specify id before the type, all objects will be numbered from scratch during parsing.

• Integer - will always occupy 4 bytes, so after the type you do not need to specify its length.

• Fractional number - will have a double representation and a length of 8 bytes, which also makes it unnecessary to specify the length.

• String - after specifying the type, you must specify the total number of bytes that this string occupies.

All this will provide the structure with the same flexibility that JSON has, but will deprive it of enormous redundancy. Now all field names are repeated once at the beginning of the structure, to determine a field in an object, you only need to indicate the id of the name, which takes only 2 bytes, its type (1 byte) and, if it is a string, array or object, the length of the value. As a result, we get that to declare a field, only 3 or 5 bytes are needed, not counting the value itself. At the same time, in JSON, to define a field, you must specify its full name, which can take unlimited length, the name must be in quotation marks, each of which takes one byte, followed by a colon, which takes another byte, and if the value is a string, an array or an object, that value should still be in quotation marks or brackets (depending on type) which will add 2 more extra bytes. As a result, we have fixed 3 or 5 bytes in the new structure, when in JSON we have a field name occupying at least 1 character (byte) and the same 3 or 5 bytes for additional formatting.

The main drawback of the structure is its complete unreadability. But since JSON is transmitted in an unformatted form, before working with it, it is formatted into a convenient structure. My structure can easily repeat JSON so that when working with it can be parsed in JSON and when passing back. This will allow people to easily work with it and transfer data with minimal redundancy.

It is also worth noting that there is a scenario in which my structure is more redundant - this is the case when the data does not have fields with the same name. However, it must be taken into account that such data, as a rule, are not large and the losses will be negligible. This structure is aimed at reducing redundancy during the transfer of large amounts of data, and these are usually arraying with repeatedly repeating objects that have a similar structure, and therefore the fields will be duplicated many times. Savings in the transfer of such structures will greatly exceed losses in the transfer of small objects, which will positively affect both the transmission speed and the amount of traffic.

REFERENCES

1. Gusfield Dan. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology - 1 edition (May 28, 1997).
2. Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation 26 November 2008 – Mode of access: https://www.w3.org/TR/xml/. - Date of access: 15.02.2020.
3. Introducing JSON – Mode of access: https://www.json.org/json-en.html. - Date of access: 15.02.2020.